Integration of Declarative and Algorithmic Approaches for Layout Creation

Tao Lin[1] and Peter Eades[2]

[1] CSIRO Division of Information Technology, Canberra, ACT 2601, Australia
[2] Department of Computer Science, The University of Newcastle, NSW 2308, Australia

Abstract. To be useful for a diagram-based interactive visualization system, a layout creation method should satisfy many requirements including user controllability, computational efficiency, flexibility and reusability. The layout created by the method should also be readable, and should conform to the semantics and syntax to the application. Unfortunately, the two traditional approaches (algorithmic and declarative) to layout creation are inadequate to meet these requirements. This paper illustrates a novel layout creation approach which integrates these two traditional approaches and satisfies the requirements mentioned above.

1 Introduction

Diagram-based interactive visualization systems (*diagram systems*) have increased in popularity in recent years. Applications include visual programming, the visualization of database schema, CASE tools, and other CAD systems (see, for example, [18]).

With such systems, users create their mental maps of the application from pictures. Diagram systems use automatic layout creation methods, that is, the system creates the layout. This not only frees users from tiresome layout tasks, but also tends to increase the quality and consistency of the layouts.

Automatic layout creation methods have received a considerable amount of attention over the past decade, and over two hundred papers are cited in [1]. On the other hand, many current diagram systems (such as the network composer in AVS and Explorer [18]) only use fairly primitive layout creation methods.

In this paper, we briefly review the reasons behind the lack of use of sophisticated automatic layout creation methods. Our work is based on the analysis, first discussed in [12], of two traditional approaches to the automatic layout creation: the algorithmic approach and the declarative approach. In this paper, we show how an integration of the traditional approaches could lead to more use of sophisticated layout methods in diagram systems.

This paper is organized as follows. In Section 2, the concepts of a Diagram User Interface (DUI) and layout are defined, and various roles in DUI construction and use are classified. Section 3 addresses the requirements of layout creation methods from the point of view of Diagram User Interfaces, and Section 4 reviews the traditional layout creation approaches. Section 5 proposes the integration of

these approaches, and Section 6 describes Tree-Snake as an illustration of this integration. Section 7 proposes future directions.

2 Layout Creation in a Diagram User Interface

For a diagram system, it is useful to separate the visual front end for handling the diagrammatic representation from the rest of the system. We call the visual front end a *Diagram User Interface (DUI)*, and the rest of the system an *application*.

Pictures in a DUI are created by graphical objects. The attributes of the graphical objects can be divided into two types: attributes such as "size", "shape", and "location" are *geometric attributes*, while attributes such as "color" and "texture" are *appearance attributes*.

Relations between graphical objects, such as "left-of" and "connected-to", are *geometric relations*. The perceptual "pattern" or "shape" is defined by the geometric attributes and relations in a picture. People naturally capture these perceptual patterns in a picture, and these patterns form the *layout* of a picture.

There are two kinds of operations in the layout process: *layout creation*, which creates the layout from scratch, and *layout adjustment*, which updates an existing layout after some manipulations. In this paper, we concentrate on layout creation methods. However, a layout creation method of DUIs should take the interaction issues into account.

In many current DUI-like systems, layout functions are created on an ad hoc basis from scratch. Since it is quite expensive to develop a layout creation method, the cost of such DUIs is very high. For a non-layout-expert designer, a toolkit for layout creation is essential for minimizing cost. At least one such toolkit is currently commercially available [15], and Bertolazzi *et al.* have produced another [2]. These toolkits perform similar function for DUIs as toolkits such as InterViews [13] perform for graphical user interfaces.

Here, we distinguish the roles in DUI construction and use as follows:

- A *DUI modeler* implements the toolkit. The DUI modelers are layout experts. Since the toolkit is not oriented towards any particular diagrammatic representations, the layout creation methods provided by the toolkit must cover a broad range of diagrammatic representations.
- A *DUI designer* develops a specific DUI for a specific application. A DUI designer implements layout creation methods using the toolkit developed by DUI modelers. DUI designers are domain experts rather than layout experts, and are only interested in finding effective layout creation methods for handling their specific diagrammatic representations.
- A *user* is the customer of a layout creation method.

3 Layout Requirements

Although layout creation methods have been available for many years, only the simplest have been used in DUIs. To understand this problem, we need to clarify

the requirements of layout creation methods for DUIs from two points of view: that of users, and that of the DUI modelers and designers.

3.1 The User Perspective

From the viewpoint of users, the requirements include:

- Ru1: *Readability.* Most of the techniques surveyed in [1] are oriented toward producing a layout which is easy to read. We use the term *readability criteria* for the requirements of readability. Some common readability criteria are:
  - small number of edge crossings,
  - symmetry, and
  - uniform distribution of vertices.
- Ru2: *Conformance.* Different applications use different diagrammatic representations. The layout should conform to the syntax and semantics of the specific diagrammatic representation handled by the DUI. Some applications have clear conformance criteria; for example, in a tree diagram which represents the hierarchy of an organization, the employer should appear above the employee. The rules which define how the layout conforms to the application are the *conformance criteria* of the DUI.
- Ru3: *Controllability.* Since a DUI is a user interface, a user should be able to control the layout to some extent. For example, a user should be able to place a particularly interesting graphical object in the center of the drawing window, or restrict the height or width of the layout (the limitation of window size). These rules imposed on a specific instantiation of a DUI are called *preference criteria.*
- Ru4: *Reasonable Response Time.* A DUI is an interactive system. Therefore, the response time of a layout creation method must not exceed the tolerance of the users. The implementation must be computationally efficient to ensure reasonable response time.

3.2 The Perspectives of Modelers and Designers

From the viewpoint of modelers and designers, the requirements include:

- Ri1: *Reusability.* Layout creation methods (especially more complex techniques) are expensive to develop and difficult to code, and thus should be reused as much as possible. Reusing layout creation methods reduces the expense of DUI design by amortizing the cost of many DUIs. Further, reuse raises quality, since the layout creation methods developed by DUI modelers who can be layout experts.
- Ri2: *Flexibility.* A layout creation method must be flexible:
  - The methods should be easily integrated with a variety of conformance criteria. A DUI designer must be able to customize the same layout creation method for use in different DUIs.
  - The methods should be adjustable at runtime to satisfy preference criteria. Thus, a DUI can also create the layout to meet the dynamic requirements of users.

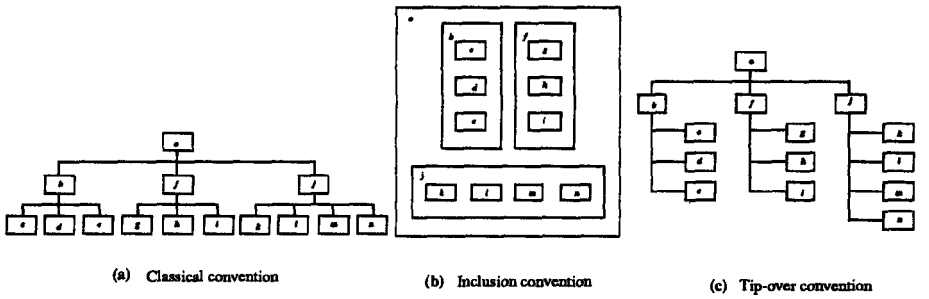

Fig. 1. Drawing conventions for rooted trees.

4 Two Traditional Approaches to Layout Creation

We can divide the current implementation approaches into two categories: those with an algorithmic approach and those with a declarative approach.

4.1 Algorithmic Approach

Many techniques surveyed in [1] take an *algorithmic approach*. The background of these techniques is in Algorithmics, Computational Geometry, and Data Structures. The developers assume well specified requirements, and attempt to construct algorithms which provably meet the requirements using minimal computational resources.

The algorithmic approach has been quite successful in achieving readability requirements (Ru1) and efficiency requirements (Ru4), and several systems use this approach, for example EDGE [14].

An algorithmic method typically operates on a specific *graph-theoretic class*, such as trees (rooted trees or free trees), planar graphs, or triangulations. Algorithmic methods can thus be classified according to the graph-theoretic classes on which they operate.

A graph-theoretic class may be further classified into a set of *drawing conventions*. For example, we can further refine the class of rooted trees according to the conventions under which they are drawn. These conventions include the "classical convention" (see Fig. 1(a)), the "inclusion convention" (see Fig. 1(b)), and the "tip-over convention" (see Fig. 1(c)) [10]. For each convention, there is a set of algorithmic methods, and each of them satisfies some readability criteria better than the others. Each algorithmic method can only be applied to certain drawing conventions, and the layout created by a single algorithmic method can only satisfy very limited readability criteria.

However, this approach has some disadvantages:

- Most algorithmic methods can only be applied to a specific drawing convention within a specific graph-theoretic class. Diagrammatic techniques often arise from pen-and-paper methods and have been established for a long time

before computerization. Because of the limited ability to deal with drawing conventions, the algorithmic approach tends to impose new diagrammatic representations in such applications. This imposition is typically resisted by people with experience in the established diagrammatic techniques.
- Layout requirements are hard coded in algorithmic methods. In general, user controllability (Ru3) and flexibility (Ri2) of algorithmic methods are severely limited.
- Algorithms typically only use graph-theoretic structure (node connection information) to compute a layout. Thus many conformance criteria which are related to the meaning of the application objects rather than the graph-theoretic structure of the relations between graphical objects are difficult to integrate.
- Some algorithmic methods (such as planarity-based techniques) are quite complex and difficult to code.

For most systems with an algorithmic approach, the DUI modeler constructs a "toolkit" of algorithmic methods; the DUI designer then chooses an appropriate method for the application. There is little or no customization, and the DUI designer is imposed to use the diagrammatic representation proposed by the algorithmic method. The user can only observe the results created by the layout creation method. In this sense, the interactive role of a user is virtually omitted.

4.2 Declarative Approach

The background of the *declarative approach* is in Artificial Intelligence. The techniques are oriented toward expressiveness, generality, and flexibility, rather than computational efficiency.

A typical system which uses a declarative method has two components: a mechanism for expressing requirements, and a mechanism for satisfying the requirements. The requirements of layout may be expressed as constraints [6, 11] or as a cost function [7]. The mechanisms for satisfying these requirements include constraint solvers [11] and simulated annealing [7].

The expressive power of constraints and rules ensures that a wide variety of layout requirements can be specified; thus the declarative approach maximizes flexibility (Ri2); this leads to the ability to conform to a wide range of applications (Ru2) and gives users good control mechanisms (Ru3). Satisfaction mechanisms such as constraint solvers are general enough (with applications extending far beyond layout creation) to ensure availability. Some mechanisms, such as simulated annealing is relatively easy to code and thus are not expensive.

However, there are some problems involved in the declarative approach:

- It is difficult to foresee the effects of a constraint or rule, even in a moderately sized system.
- Typical declarative methods are not efficient, even with a reasonably small number of constraints or rules. This leads to excessive response times (Ru4).

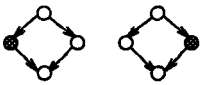

Fig. 2. Nondeterminism in an algorithmic method.

- The mechanisms for creating layout are usually heuristic and it is impossible to guarantee results. Sometimes declarative methods do not achieve their aim, even if there were a layout which satisfies relevant layout requirements.
- Some declarative methods have difficulty handling readability criteria (Ru1), especially those with a global nature, such as planarity.

The roles mention in Sect. 2 are easily defined in the declarative approach. The DUI modeler provides mechanisms for expressing and satisfying requirements. The DUI designer can easily develop a layout creation method by expressing requirements according to conformance criteria. Further, a user can control the layout by expressing some preference criteria (Ru3), even at run time. Unfortunately, the problems listed above are difficult to overcome with a declarative approach.

5 The Integrated Approach

From the analysis above, we conclude that neither algorithmic nor declarative approaches are sufficient for sole adoption in a DUI. By integrating the approaches, the advantages of one may be used to compensate for the disadvantages of the other.

If an algorithmic method is implemented on top of a declarative system, the integration can be as slow as the declarative system. Therefore, the integration should be implemented by placing the declarative mechanisms on top of an algorithmic method.

Many algorithmic methods are nondeterministic, in the sense that the precise output is not determined from the input. For instance, using a Sugiyama method (see, for example, [17]), either of Fig. 2(a) or (b) could be produced. Other methods which include some nondeterminism are planarity-based methods, bend minimization methods, most tree layout algorithms, and some visibility methods.

This nondeterministic behavior provides *options* which can be given values according to conformance criteria, or preference criteria.

To integrate declarative mechanisms with an algorithmic method, we can use declarative mechanisms to specify further restrictions on the options, and to adjust the layout to follow the restrictions. We call a method using such integration a *customizable method.*

A customizable method is based on an algorithmic method, and can perform limited adjustments to the algorithmic method. Therefore, a customizable method can only satisfy certain readability criteria, conformance and preference criteria as well.

Thus the role of a DUI modeler is to create a toolkit of customizable methods. To implement a layout creation method for an application, a DUI designer specifies the conformance criteria using the options. A user further specifies preference criteria using the options.

Several existing systems adopt approaches which are integrated to some extent. Extended EDGE [4] integrates some closeness constraints with the Sugiyama method [17]. These constraints are mainly aimed at local "mental map" preservation; for example, the constraints maintain geometric "clusters" of nodes after a layout modification. Extended ANDD [8] integrates some constraints defining "Visual Organization Features" with the "spring" algorithm [9]. The constraints preserve features such as the "left-right order", "alignment", and some special "shapes" for a set of graphical objects.

Some algorithms have been developed with a view to customization; for an example, see [3].

6 Tree-Snake

This section illustrates the approach described in the last section with a customizable method for drawing trees in the Tree-Snake [12] system.

First, we review an algorithmic method for drawing binary trees according to a specific convention; the method is from [10]. Then, we show how this algorithmic method may be extended to a customizable method, and illustrate the use of the customizable method.

6.1 Inclusion Convention Binary Tree Drawing

Suppose that *tree* $T = (V, A, r)$ is a rooted binary tree consisting of a set V of *nodes*, a *root* r, and an *edge* set $A \subseteq V \times V$. An *inclusion layout* (an example is shown in Figure 1(b)) for a tree $T = (V, A, r)$ consists of a rectangle R_u in the plane for each node u of T, such that

- if u has a child w then R_w is within R_u, and
- if u has children v and w then the rectangles R_v and R_w do not overlap and are separated by a distance of at least p.

The **Minimum Inclusion Layout Problem** (MILP) is as follows:

> Given a tree $T = (V, A, r)$ and a width X_v and height Y_v for each leaf v of T, find a minimum size (minimum bounding box for T) inclusion layout for T such that for each leaf v, the dimensions of R_v are $X_v \times Y_v$.

The key idea to solve MILP is that if u has children v and w, then there are essentially only two ways of arranging R_v and R_w within R_u: either vertically, or horizontally, as in Figure 3.

Note that MILP is similar to the slicing layout problem in VLSI [16]. Several algorithms for MILP are given in [10]. In particular, a dynamic programming

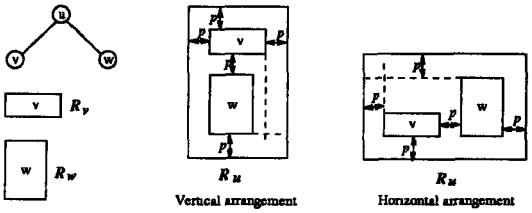

Fig. 3. Vertical and horizontal arrangements.

approach is presented. If the input sizes X_v, Y_v for each leaf v are determined by text lying within the nodes, then an optimal layout can be found in time $O(nM)$ where n is the number of nodes and M total number of characters of text. This result has been extended for n-ary tree *tip-over convention* (an example is shown in Figure 1(c)). For details see [10].

6.2 Customizable Methods in TreeSnake

We have developed a customizable method based on the algorithmic method mentioned above.

Options are shown as follows:

- *Order relations* (left-right or top-down) on a set of siblings can be specified by constraints.
- The *orientation* of a subtree can be specified: it can be forced to be vertical, or horizontal, or can be left to the algorithm to decide.
- The *size* of a subtree can be restricted by height, width, or area (height $\times$ width)).
- *Margins* (the space between graphical objects) can be set.
- *Ports* can be specified (that is, the method to connect two graphical objects).
- The *image* of objects (the graphical object for either node or edge) can be specified.

A customizable method was created based on the integration of the capacity for handling these options and the algorithmic method mentioned above. The first option is implemented with a simple extension of topological sorting techniques; see [12]. The second and third options were implemented by adjusting the dynamic programming approach to MILP [10]. The other options are simple parameters which can also be controlled.

By specifying conformance criteria, this customizable method has been applied to give several layout creation methods used in different DUIs as shown in Figure 4.

- At the top left is a DUI for handling list diagrams [11], and may be used as a visualization system for LISP. The diagrammatic representation here is essentially the *hv-tree*, from [5].

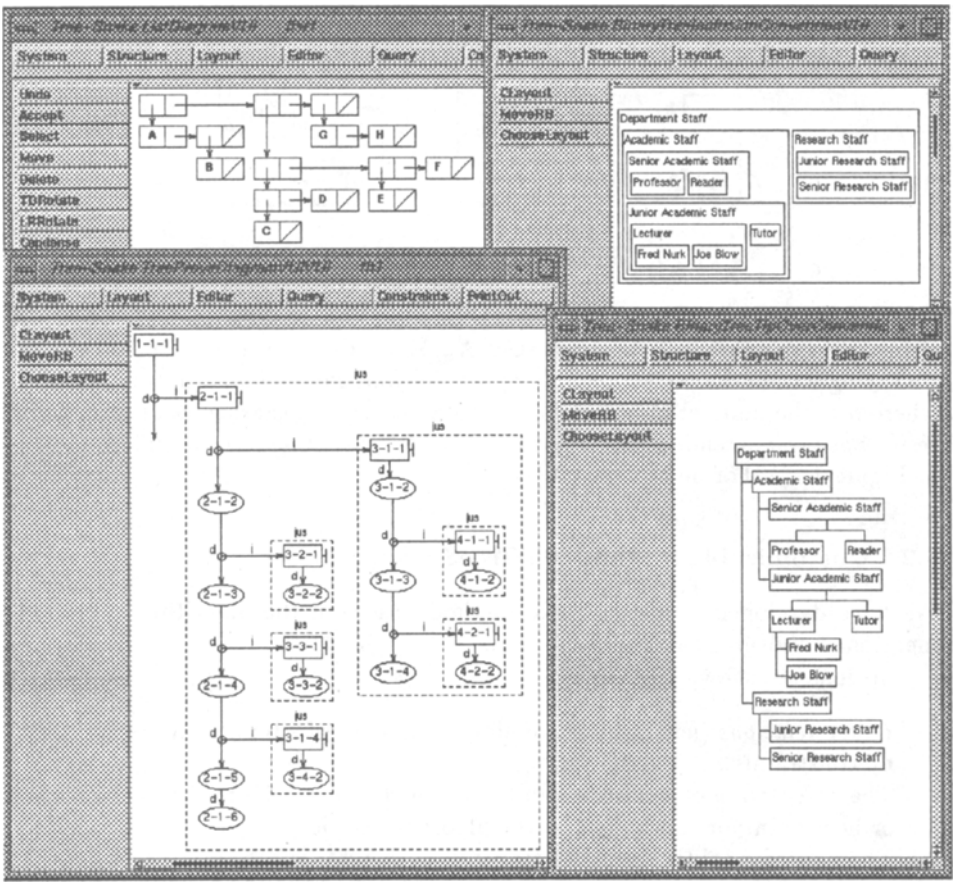

Fig. 4. Layouts created by the same customizable method.

- At the top right and left bottom illustrate two DUIs for handling organization charts. These two DUIs use different diagrammatic representations: the top right uses an inclusion layout or treemap. The bottom left uses tip-over trees. These DUIs are described more fully in [12].
- At the bottom left is a DUI for handling theorem proving diagrams; this is used in an interactive theorem proving system [12]. The nodes represent theorems and the edges are dependencies between the theorems. Note that the diagrammatic representation here is similar to the hv-trees.

Conformance criteria were specified to create each of these four DUIs from a single customizable method.

Within each of these DUIs, a user can specify preference criteria to control the layout. For instance, a user can constrain the diagram to a given width, and give a specific position to a specific node. As an example, consider the organization chart in the top right of Figure 4, and compare with Figure 5. The diagram in

Figure 5 is optimally compact in terms of area; but by specifying a maximum height, the user obtains the top right of Figure 4; this has minimum area subject to the height constraint.

The Tree-Snake system is implemented in ParcPlace Smalltalk (using VisualWorks) on a SparcStation. Conformance criteria are treated as class variables which are applied to all the object instances in that class, and preference criteria as instance variables which are only applied to specific object instance. For example, the height restriction discussed above only applies to root node ("Department Staff") in the case shown in Figure 5, and not to the other instances of the node class.

While using the layout creation method, the consistency between the new criterion value and the conformance criteria is checked before a new preference criterion is applied. If the new preference criterion conflicts with conformance criteria, the system assumes that the user intended to modify the application. For example, suppose that a user changes the left-right order of a set of siblings. If the conformance criteria prohibit this permutation, then the system may not only change the layout, but also the associated objects in the application[3]. If new preference criteria conflicts with the existing preference criteria, then the DUI deletes the existing preference criteria in conflict.

It can be shown [12] that subject to the constraints imposed by the customization according to the options mentioned above, the layout creation methods derived from this customizable give drawings which are optimally compact; this is the main readability criterion (Ru1) for trees. Also, it can be shown [12] that the methods run in the same time complexity ($O(nM)$) as the algorithmic method for MILP; that is, they are efficient (Ru4). Thus we have all the advantages of the algorithmic approach.

Further, the layout creation methods derived from the customizable method offer a reasonable amount of flexibility (Ri2) for various conformance (Ru2) and preference (Ru3) criteria. A DUI designer can specify the options in this customizable method to meet conformance criteria without coding, and a user can specify the options to meet preference criteria to control the layout. This customization power increases reusability (Ri1) and thus reduces the expense to the development of DUIs. Thus it has much of the advantages of the declarative approach.

7 Conclusions and Future Work

We have analyzed the requirements of layout creation methods for diagram user interfaces, and concluded that traditional approaches cannot satisfy these requirements. This paper proposes a novel approach which integrates the traditional approaches. We believe that this overcomes many of the deficiencies of traditional approaches, while maintaining their advantages.

[3] Techniques of constraint maintenance can be quite subtle; for more detail see [12].

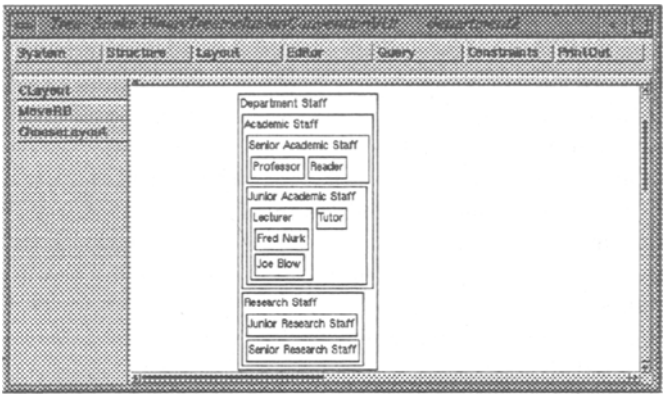

Fig. 5. The optimal compact layout in inclusion convention.

A layout creation method based on integrated approach is a customizable method. With a customizable method developed by a DUI modeler, a DUI designer can specify the options to meet the conformance criteria, and the users can further specify the options to meet their preference criteria.

We have illustrated this in Tree-Snake where a customizable method has been applied to handle four different diagrammatic representations, without coding.

The example shown here is a very simple case. To have an efficient integrated method for some complicated case (for example, to combine complex constraints with a planarity-based algorithm), further work is needed.

We also note that there are some problems for the use of our approach:

1. It is difficult to specify the constraints with text.
2. Each customizable method can only satisfy limited types of conformance and preference criteria.

For the first problem, we believe that the use of visual constraint specification systems (such as DOODLE [6]) provide good mechanisms to make graphical specifications on diagrammatic representations. We plan to investigate the combination of visual specification with our integrated approach.

The only reasonable solution for the second problem is to provide a set, or toolkit, of customizable methods for layout creation. Such a toolkit should cover broad readability, conformance and preference criteria. Bertolazzi *et.al* have developed ALF [2], currently the most extensive toolkit of algorithmic methods. ALF is oriented mostly towards readability criteria, and the other requirements discussed in Sect. 3 have not been considered. However, ALF and the *Diagram Server* (the environment of ALF) provides many useful concepts for the development of a toolkit of customizable methods. We believe that such a toolkit built around the structure and philosophy of ALF would be a significant contribution.

Acknowledgement

We would like to thank Matthew O. Ward and Kenneth Tsui who read this paper, and gave many useful suggestions.

References

1. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. Algorithms for drawing graphs: An annotated bibliography. To appear in Computational Geometry and Applications.
2. P. Bertolazzi, G. Di Battista, and G. Liotta. Parametric graph drawing. Technical Report 6/67, Dipartmento di Informatica e Sistemistica, Universita' di Room "La Sapienza", September 1992.
3. G. Di Battista, R. Tamassia, and I.G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, 41:1–7, 1992.
4. K. Bohringer and F. Newbery Paulisch. Using constraints to acheive stability in automatic graph layout algorithms. In *Proceedings of ACM CHI 90*, pages 43–51., 1990.
5. P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. Technical Report 11.91, Dipartimento di Informatica e Sistemistica, Univ. di Roma "La Sapienza", 1991.
6. I. Cruz. Expressing constraints for data display specification: a visual approach. Technical Report CS-93-57, Brown University, 1993.
7. R. Davidson and D. Harel. Drawing graphs nicely using simulated aneealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.
8. E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Visual Languages 93*, 1993.
9. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
10. P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. *International Journal of Computational Geometry and Applications*, 3(2):133 - 153, 1993.
11. T. Kamada. *Visualizing Abstract Objects and Relations*. World Scientific Series in Computer Science, 1989.
12. T. Lin. *Diagram User Interfaces*. PhD thesis, University of Newcastle. 1993.
13. M. Linton, J. Vlissides, and P. Calder. Composing user interfaces with interviews. *Computer*, 22(2):8 - 22, 1989.
14. F. Newbery Paulish and W.F. Tichy. Edge: An extendible graph editor. *Software – Practice and Experience*, 20(S1):1/63–S1/88, 1990. also as Technical Report 8/88, Fakultat fur Informatik, Univ. of Karlsruhe, 1988.
15. Tom Sawyer Software. Graph layout toolkit. available from bmadden@TomSawyer.COM.
16. L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57:91 - 101, 1983.
17. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(2):109–125, 1981.
18. C. Williams, J. Rasure, and C. Hansen. The state of the art of visual langauges for visualization. In *Visualization 92*, pages 202 - 209, 1992.